Ficha 1

Programação Imperativa

1 Estado e atribuições

Diga, justificando, qual o output de cada um dos seguintes excertos de código C.

1.

```
int x, y;
x = 3;    y = x+1;
x = x*y; y = x + y;
printf("%d %d\n", x, y);
```

2.

```
int x, y;
x = 0;
printf ("%d %d\n", x, y);
```

3. (assuma que os códigos ASCII dos caracteres `'A'`, `'0'`, `' '` e `'a'` são respectivamente 65, 48, 32 e 97)

```
char a, b, c;
a = 'A'; b = ' '; c = '0';
printf ("%c %d\n", a, a);
a = a+1; c = c+2;
printf ("%c %d %c %d\n", a, a, c, c);
c = a + b;
printf ("%c %d\n", c, c);
```

4.

```
int x, y;
x = 200; y = 100;
x = x+y; y = x-y; x = x-y;
printf ("%d %d\n", x, y);
```

5.

```
int x, y;
x = 100; y = 28;
x += y ; y -= x ;
printf ("%d %d\n", x++, ++y);
printf ("%d %d\n", x, y);
```

2 Estruturas de controlo

1. Diga, justificando, qual o output de cada um dos seguintes excertos de código C.

   (a)

   ```
   int x, y;
   x = 3; y = 5;
   if (x > y)
    y = 6;
   printf ("%d %d\n", x, y);
   ```

   (b)

   ```
   int x, y;
   x = y = 0;
   while (x != 11) {
     x = x+1; y += x;
   }
   printf ("%d %d\n", x, y);
   ```

   (c)

   ```
   int x, y;
   x = y = 0;
   while (x != 11) {
     x = x+2; y += x;
   }
   printf ("%d %d\n", x, y);
   ```

   (d)

   ```
   int i;
   for (i=0; (i<20) ; i++)
     if (i%2 == 0) putchar ('_');
     else putchar ('#');
   ```

   (e)

   ```
   char i, j;
   for (i='a'; (i != 'h'); i++) {
     for (j=i; (j != 'h'); j++)
       putchar (j);
     putchar ('\n');
   }
   ```

   (f)

   ```
   void f (int n) {
     while (n>0) {
       if (n%2 == 0) putchar ('0');
       else putchar ('1');
   ```

```c
      n = n/2;
    }
    putchar ('\n');
  }
  int main () {
    int i;
    for (i=0;(i<16);i++)
      f (i);
    return 0;
  }
```

2. Escreva um programa que desenhe no ecran (usando o caracter `#`) um quadrado de dimensão 5. Defina para isso uma função que desenha um quadrado de dimensão `n`. Use a função `putchar`. O resultado da invocação dessa função com um argumento `5` deverá ser

```
#####
#####
#####
#####
#####
```

3. Escreva um programa que desenhe no ecran (usando os caracteres `#` e `_`) um tabuleiro de xadrez. Defina para isso uma função que desenha um tabuleiro de xadrez de dimensão `n`. Use a função `putchar`. O resultado da invocação dessa função com um argumento `5` deverá ser

```
#_#_#
_#_#_
#_#_#
_#_#_
#_#_#
```

4. Escreva duas funções que desenham triangulos (usando o caracter `#`). O resultado da invocação dessas funções com um argumento `5` deverá ser

```
#
##
###                               #
####                             ###
#####                           #####
####                           #######
###                           #########
##
#
```

Defina cada uma dessas funções (com o nome `triangulo`), num ficheiro separado (`vertical.c` e `horizontal.c`). Compile esses dois ficheiros (usando o comando `gcc -c`) separadamente.

Considere agora o seguinte programa `triangulo.c`

```c
#include<stdio.h>

void triangulo (int n);

main () {
  triangulo (5);
  return 0;
}
```

Compile este programa (com o comando `gcc -c triangulo.c`). Construa (e use) agora dois executáveis, usando os comandos

- `gcc -o t1 triangulo.o vertical.o`
- `gcc -o t2 triangulo.o horizontal.o`

3 Memória e endereçamento

1. Diga justificando qual o resultado de executar o seguinte programa:

   ```c
   int main () {
     int i, j, *a, *b;

     i=3; j=5;
     a = &i; b = &j;
     i++;
     j = i + *b;
     b = a;
     j = j + *b;
     printf ("%d\n", j);

     return 0;
   }
   ```

2. Considere a seguinte definição de uma função `void init (int a)`

   ```c
   void init (int a) {
    a = 0;
   }
   ```

   Diga justificando qual o resultado de executar o seguinte código:

   ```c
   int x;
   x = 3;
   init (x);
   printf("%d\n", x);
   ```

   Como modificaria a função (e a sua invocação) para que o resultado fosse `0`.

3. Defina uma função `void swap (.....)` que troca o valor de duas variáveis. Por exemplo, o código

```
int x = 3, y = 5;
swap(...);
printf ("%d␣%d\n", x, y);
```

deverá imprimir no ecran `5 3`.

4 Algoritmos Numéricos sobre inteiros

1. Uma forma de definir a multiplicação por um inteiro é através de um somatório de parcelas constantes.

   Assim
   $$n \times m = \sum_{i=1}^{n} m$$
   Esta definição corresponde à definição recursiva que se apresenta à direita.
   Apresente ums definição iterativa desta função.

   ```
   int mult (int n, int m) {
    int r;
    if (n>0)
      r = m + mult (n-1, m);
    else r = 0;

    return r;
   }
   ```

2. Uma forma alternativa (e muito mais eficiente) consiste em aproveitar a representação binária dos inteiros (onde a multiplicação e divisão por 2 são pelo menos tão eficientes como a adição).

   Se analisarmos a definição anterior em dois casos (caso em que o multiplicador é par ou ímpar), obtemos a seguinte definição:

   $$n \times m = \begin{cases} 0 & \text{Se } n = 0 \\ n/2 \times (m * 2) & \text{Se } n \text{ é par} \\ m + ((n-1)/2 \times (m * 2)) & \text{Se } n \text{ é ímpar} \end{cases}$$

   Que corresponde à definição recursiva que se apresenta abaixo.

   Apresente ums definição iterativa desta função.

   ```
   int mult (int n, int m) {
     int r = 0;
     if (n>0) {
       if (n % 2 != 0)
         r = r + m;
       r = r + mult (n>>1, m<<1);
     }
     return r;
   }
   ```

3. O cálculo do máximo divisor comum entre dois números inteiros não negativos pode ser feito, de uma forma muito pouco eficiente, procurando de entre os divisores do menor deles, o maior que é também divisor do outro.
Quantas iterações faz o ciclo desta função para valores dos argumentos de `(1,1000)` e `(999,1000)`?

```
int mdc (int a, int b) {
  int d;

  if (a>b) d = b;
  else     d = a;

  while ((a % d != 0) ||
         (b % d != 0));
    d--;

  return d;
}
```

4. Uma forma alternativa de calcular o máximo divisor comum (mdc) baseia-se na seguinte propriedade demonstrada por Euclides: para a e b inteiros positivos,

$$mdc\ (a, b) = mdc\ (a + b, b)$$

Desta propriedade podemos concluir que:

$$\text{mdc } (a,b) = \begin{cases} mdc\ (a-b, b) & \text{Se } a > b \\ mdc\ (a, b-a) & \text{Se } a < b \\ a & \text{Se } a = b \end{cases}$$

Que corresponde à definição recursiva que se apresenta à direita.
Apresente ums definição iterativa desta função.

```
int mdc (int a, int b) {
  int r;
  if (a == b) r = a;
  else if (a > b)
    r = mdc (a-b, b);
  else r = mdc (a, b-a);
  return r;
}
```

5. Quantas iterações faz o ciclo da função que apresentou na alínea anterior para valores dos argumentos de `(1,1000)` e `(999,1000)`?

6. Uma forma de melhorar o comportamento do algoritmo de Euclides consiste em substituir as operações de subtracção por operações de `%` (resto da divisão inteira). Repita o exercício da alínea anterior para essa variante do algoritmo.

7. Uma outra variante do algoritmo de Euclides para o cálculo do mdc, é conhecida como o algoritmo de Euclides binário, e tal como já vimos noutros casos, usa o facto de a multiplicação e divisão por 2 serem operações muito eficientes.

   Os seguintes passos[1] para calcular `mdc (a,b)` descrevem esta variante:

   (a) `mdc (0,a)` = `mdc (a,0)` = `a`
   (b) Se tanto `a` como `b` forem pares, `mdc(a,b)` = `2*mdc(a/2, b/2)`
   (c) Se `a` for par e `b` ímpar, `mdc(a,b)` = `mdc(a/2,b)`
   (d) Se `a` for ímpar e `b` par, `mdc(a,b)` = `mdc(a,b/2)`

[1] ver em `en.wikipedia.org/wiki/Binary_GCD_algorithm`

(e) Se ambos forem ímpares e `a >= b`, `mdc(a,b)` = `mdc ((a-b)/2,b)`

(f) Se ambos forem ímpares e `a <= b`, `mdc(a,b)` = `mdc (a,(b-a)/2)`

O algoritmo continua a por aplicar estes passos até que um dos argumentos seja `0`. O resultado obtido deverá então ser multiplicado por 2 tantas vezes quantas tiver sido aplicado passo 7b.

Codifique este algoritmo em C, tendo o cuidado de usar as operações `<<` e `>>` para multiplicar e dividir por 2.

8. A sequência de Fibonacci define-se como

$$fib\ (n) = \begin{cases} 1 & \text{Se } n < 2 \\ fib\ (n-1) + fib\ (n-2) & \text{Se } n \geq 2 \end{cases}$$

(a) Apresente uma definição recursiva de uma função que calcula o n-ésimo número desta sequência.

(b) O cálculo do n-ésimo número de Fibonacci pode ser definido de uma forma mais eficiente (e iterativa) se repararmos que ele apenas necessita de conhecer os valores dos 2 valores anteriores. Apresente uma definição alternativa (e iterativa) da função da alínea anterior que calcula o n-ésimo número de Fibonacci, usando duas variáveis auxiliares que guardam os dois valores anteriores.

9. O complemento para dois de um inteiro pode ser calculado de duas formas:

- adicionando `1` ao complemento para um desse número (e este número resulta de inverter todos os bits de um número). Para calcular o complemento para dois de `100100100` começamos por calcular o seu complemento para um `011011011` e depois adicionamos `1` `011011100`
- aplicando o seguinte algoritmo (considerando os bits da direita para a esquerda, i.e., do menos significativo para o mais significativo), por exemplo para o número `100100100`:
  (a) enquanto forem `0`, preservam-se (obtendo `1001001 00`)
  (b) o primeiro `1` também é preservado (obtendo `100100 100`)
  (c) todos os restantes são invertidos (obtendo `011011100`).

Apresente duas definições para o cálculo do complemento para dois de um inteiro, baseadas nas alternativas apresentadas.